Thực hành 3

Đinh Minh Phương 21010604 – Phạm Anh Tuấn 21011117

- Vi Đăng Quang 21010583 - Nguyễn Quang Linh 21010581

- Nguyễn Mạnh Cường 21011583

**Bài 1:** Cho xâu: aa + a\*

**Suy dẫn trái cho chuỗi:**

|  |  |
| --- | --- |
| S → S S \* | # S được thay thế bởi S S \* |
| → SS + S\* | # S được thay thế bởi SS+ |
| → a S + S\* | # S được thay thế bởi a |
| → a a + S \* | # S được thay thế bởi a |
| → a a + a \* | # S được thay thế bởi a |

**Suy dẫn phải cho chuỗi:**

|  |  |
| --- | --- |
| S → S S \* | # S được thay thế bởi S S \* |
| → S a \* | # S được thay thế bởi a |
| → S S + a \* | # S được thay thế bởi SS+ |
| → S a + a \* | # S được thay thế bởi a \* |
| → aa + a\* | # S được thay thế bởi a |

**Cây phân tích cú pháp:**

S

/ | \

S + S

/ \ |

a S \*

/ \

a \*

|

Ε

Là văn phạm này là **rõ ràng** vì mỗi ký tự trong xâu đều có duy nhất một cách suy dẫn.

Ngôn ngữ được tạo ra bởi văn phạm này là tập hợp các xâu được tạo ra bằng cách lặp lại chuỗi aa và thêm một hoặc nhiều ký tự + vào giữa các chuỗi aa, hoặc thêm một hoặc nhiều ký tự \* vào cuối chuỗi aa.

**Bài 3:**

a) Ngôn ngữ được sinh bởi văn phạm G có đặc điểm là ngôn ngữ này chứa các chuỗi có dạng "a" và "b" được tạo ra bằng cách lặp lại việc thêm "a" vào giữa các "b" trong chuỗi.

b) Mọi phương án suy từ S => \* babab:

1. S => aS => aaS => aabS => aabaS => aabaS => aababaS => aababa

2. S => aS => aabS => aababS => aababaS => aababab

3. S => bS => baS => babS => babaS => babab

Công thứ tổng quát tính số lượng suy dẫn từ S thành chuỗi (ba)\*b là trong đó n là số lần lặp của quy tắc S => aS.

**Bài 4:**

Exp → Exp + Exp

Exp → Exp / Exp

Exp → số

Exp → (Exp)

|  |  |
| --- | --- |
| a)  Exp → Exp / Exp  |  → Exp / (Exp)  |  → Exp / (Exp + Exp) | b)  Exp → Exp / Exp  → ( Exp) / (Exp)  → (Exp + Exp) / (Exp + Exp) |
| c)  Exp → Exp + Exp  |  → Exp / Exp + Exp | d)  Exp → Exp + Exp  |  → Exp + Exp + Exp |

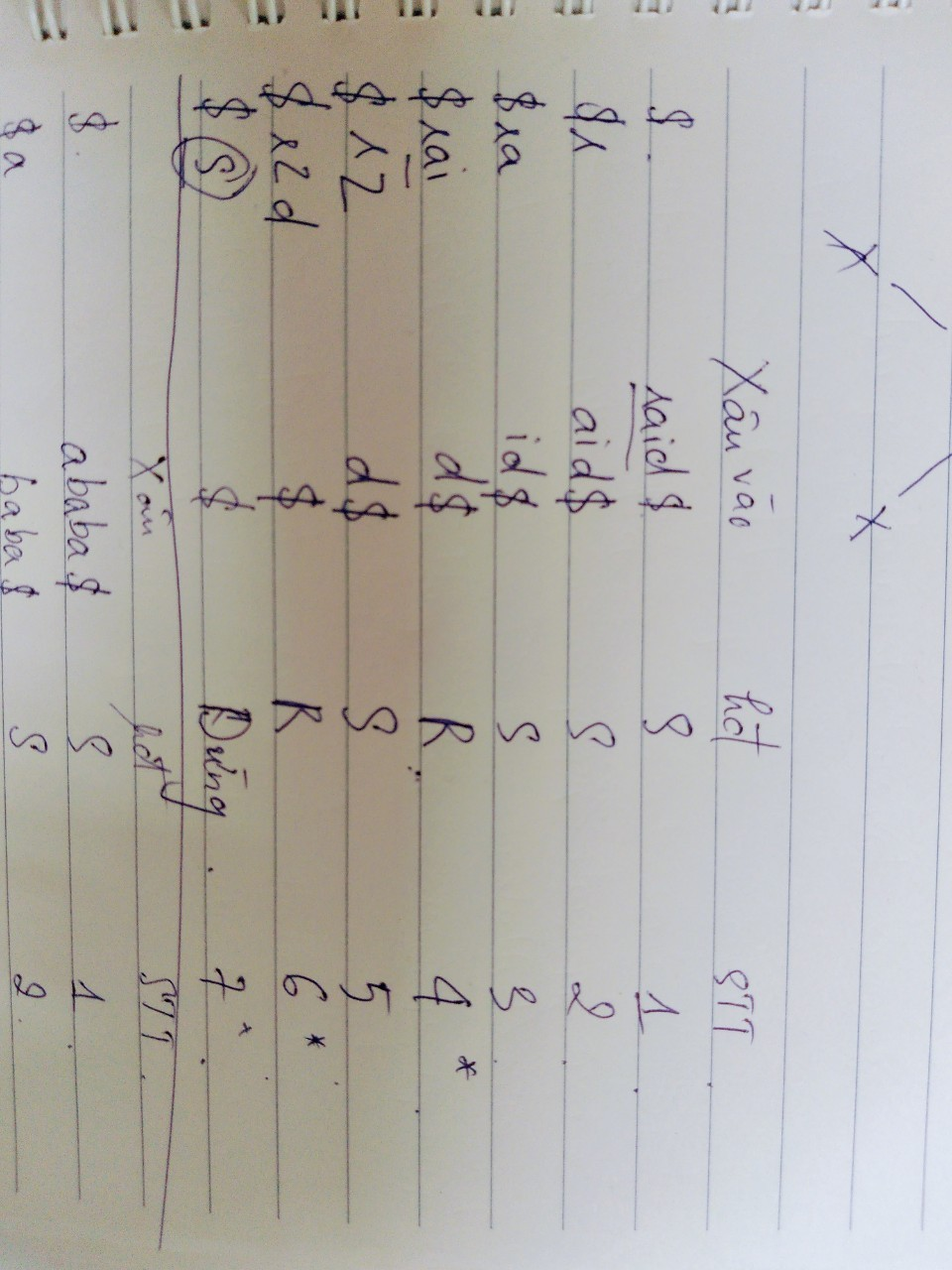
e)

Exp → Exp + Exp

|

→ Exp + Exp / Exp

**Bài 5:**

****

**Bài 6:**

Phân tích top-down chuỗi ((x+y)=(y+x)) thuộc văn phạm G

Bước 1: Khởi tạo cây phân tích với nút gốc S.

Bước 2: Áp dụng quy tắc S → B cho nút gốc S.

Bước 3: Áp dụng quy tắc B → ( B ) cho nút B vừa tạo.

Bước 4: Áp dụng quy tắc B → R cho nút B bên trong ngoặc.

Bước 5: Áp dụng quy tắc R → E = E cho nút R.

Bước 6: Áp dụng quy tắc E → x cho nút E bên trái dấu =.

Bước 7: Áp dụng quy tắc E → y cho nút E bên phải dấu =.

Bước 8: Áp dụng quy tắc E → ( E + E ) cho nút E bên trong ngoặc.

Bước 9: Áp dụng quy tắc E → x cho nút E bên trái dấu +.

Bước 10: Áp dụng quy tắc E → y cho nút E bên phải dấu +.

Kết quả: Cây phân tích hoàn chỉnh cho chuỗi ((x+y)=(y+x)) như sau:

S

/ \

B B

/ \ / \

( ) R

/ \

E =

/ \ / \

x y E E

/ \

x y

**Bài 7:**

|  |
| --- |
| function parse\_S(input\_str):  global current\_index  if input\_str[current\_index] == '+':  current\_index += 1  parse\_S(input\_str)  parse\_S(input\_str)  elif input\_str[current\_index] == '-':  current\_index += 1  parse\_S(input\_str)  parse\_S(input\_str)  elif input\_str[current\_index] == 'a':  current\_index += 1  else:  # Xử lý lỗi hoặc thông báo về ký tự không hợp lệ  print("Unexpected input at index", current\_index)  # Hàm main để bắt đầu phân tích cú pháp  function main(input\_str):  global current\_index  current\_index = 0  parse\_S(input\_str) |

**Bài 8:**

|  |
| --- |
| def parse(input\_str):  # Chuyển đổi chuỗi đầu vào thành danh sách các ký tự để dễ quản lý  tokens = list(input\_str)  # Biến index để theo dõi vị trí hiện tại trong danh sách các ký tự  index = 0  # Hàm so khớp ký tự hiện tại với ký tự được truyền vào  def match(symbol):  nonlocal index  if index < len(tokens) and tokens[index] == symbol:  index += 1  return True  return False  # Hàm đệ quy thực hiện phân tích cú pháp theo quy tắc S -> S ( S ) S | ε  def S():  nonlocal index  if match('('):  S()  if match(')'):  S()  else:  raise SyntaxError("Dự kiến có dấu ngoặc đóng")  else:  pass # ε (epsilon), không làm gì cả  try:  # Bắt đầu quá trình phân tích cú pháp bằng cách gọi hàm S  S()  # Kiểm tra xem chúng ta đã xử lý hết tất cả các ký tự hay không  if index == len(tokens):  print("Phân tích cú pháp thành công")  else:  print("Phân tích cú pháp thất bại")  except SyntaxError as e:  print(f"Phân tích cú pháp thất bại: có kí tự khác tập không gian") |

|  |
| --- |
| char input\_str[100];  int current\_index;  void consume(char expected\_char) {  if (input\_str[current\_index] == expected\_char) {  current\_index++;  } else {  printf("Expected '%c' but found '%c' at index %d\n", expected\_char, input\_str[current\_index], current\_index);  }  }  void parse\_S() {  if (input\_str[current\_index] == '0') {  consume('0');  parse\_S();  consume('1');  } else if (input\_str[current\_index] == '(') {  consume('(');  parse\_S();  consume(')');  parse\_S();  }  }  int main() {  current\_index = 0;  // Gán giá trị cho biến input\_str với đầu vào bất kỳ  strcpy(input\_str, "000111");  parse\_S();  if (input\_str[current\_index] == '\0') {  printf("Parsing successful\n");  } else {  printf("Unexpected input at index %d\n", current\_index);  }  return 0;  } |

**Bài 9:**

|  |  |
| --- | --- |
| Bài 7 | #include <iostream>  #include <string>  bool match\_terminal(char symbol, int current\_index, const std::string& input\_str) {  if (current\_index < input\_str.length() && input\_str[current\_index] == symbol) {  return true;  }  return false;  }  std::pair<bool, int> S(int current\_index, const std::string& input\_str);  std::pair<bool, int> S(int current\_index, const std::string& input\_str) {  if (current\_index < input\_str.length()) {  bool result = match\_terminal('a', current\_index, input\_str);  if (result) {  return std::make\_pair(true, current\_index + 1);  } else if (current\_index < input\_str.length()) {  if (input\_str[current\_index] == '+') {  auto [result1, current\_index1] = S(current\_index + 1, input\_str);  auto [result2, current\_index2] = S(current\_index1, input\_str);  if (result1 && result2) {  return std::make\_pair(true, current\_index2);  }  } else if (input\_str[current\_index] == '-') {  auto [result1, current\_index1] = S(current\_index + 1, input\_str);  auto [result2, current\_index2] = S(current\_index1, input\_str);  if (result1 && result2) {  return std::make\_pair(true, current\_index2);  }  }  }  }  return std::make\_pair(false, current\_index);  }  void parse(const std::string& input\_str) {  // Bắt đầu phân tích cú pháp từ ký tự đầu tiên  auto [result, current\_index] = S(0, input\_str);  if (result && current\_index == input\_str.length()) {  std::cout << "Chuoi dau vao duoc chap nhan." << std::endl;  } else {  std::cout << "Chuoi dau vao khong hop le." << std::endl;  }  }  int main() {  // Thử nghiệm với một đầu vào bất kỳ  std::string input\_str = "+++++++aaaaaaaa";  parse(input\_str);  return 0;  } |
| Bài 8 | #include <iostream>  #include <string>  class Parser {  public:  Parser(const std::string& input\_str) : input\_str(input\_str), index(0) {}  void parse() {  try {  S();  if (index == input\_str.length()) {  std::cout << "Phân tích cú pháp thành công" << std::endl;  } else {  std::cout << "Phân tích cú pháp thất bại" << std::endl;  }  } catch (const std::runtime\_error& e) {  std::cout << "Phân tích cú pháp thất bại:có chứa kí tự nằm ngoài tập không gian mẫu ";  }  }  private:  void S() {  if (match('(')) {  S();  if (match(')')) {  S();  } else {  throw std::runtime\_error("Dự kiến có dấu ngoặc đóng");  }  } else {  // ε (epsilon), không làm gì cả  }  }  bool match(char symbol) {  if (index < input\_str.length() && input\_str[index] == symbol) {  index++;  return true;  }  return false;  }  private:  const std::string& input\_str;  size\_t index;  };  int main() {  // Thử nghiệm với một đầu vào bất kỳ  std::string input\_str = "()(()))";  Parser parser(input\_str);  parser.parse();  return 0;  } |